CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

CE143: COMPUTER CONCEPTS & PROGRAMMING

Chapter – 7

Arrays

DEPSTAR

Devang Patel Institute of Advance Technology And Research

Objectives

- To learn about the necessity of using arrays
- To Examine Arrays of various dimensions
- To discover how memory allocation is done for arrays compared to other data

Introduction

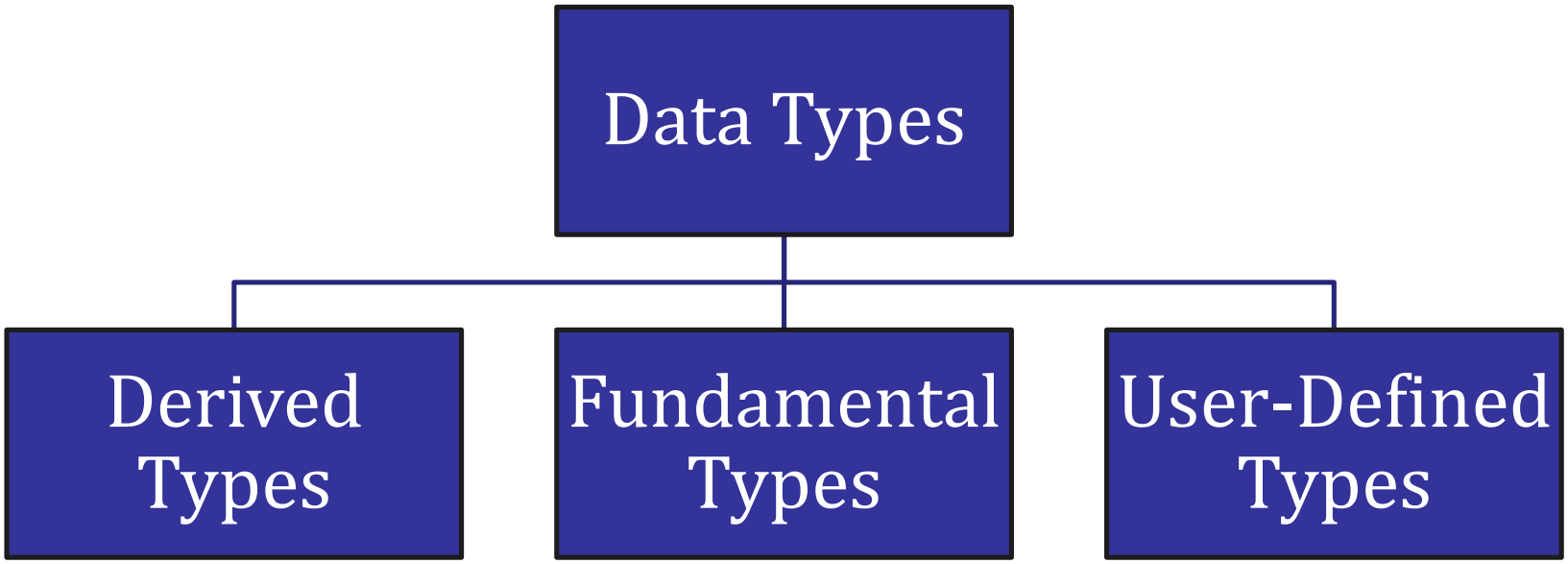
- In this chapter, we will discuss
  - Need of arrays
  - Declaration & Initialization 1D arrays
  - Programs of 1D arrays
  - 2D arrays
  - Memory allocation of 1D and 2D arrays
  - 2D array basic programs.

Need of Arrays

- Consider a situation where we need to store five integer numbers. If we use programming's simple variable and data type concepts, then we need five variables of **int** data type.

- It is simple, because we had to store just five integer numbers. Now let's assume we have to store 5000 integer numbers. Are we going to use 5000 variables?

- To handle such situations, almost all the programming languages provide a concept called **array**.


Data Types
Derived Types
Fundamental Types
User-Defined Types
Arrays
Functions
Pointers
Integral Types
Float Types
Character Types
Structures
Unions
Enumerations

Arrays

- An array is a data structure, which can store a fixed-size collection of elements of the same data type.

- Simply saying, an array is a collection of variables of the same type.

- Instead of declaring individual variables, such as var1, var2, ..., var99, var100 you just declare one array variable **var[100]** and address individual elements using an index like var[0], var[1], ..., var[98], var[99].

Declaration & Initialization 1D arrays

- Syntax:

  <type> <arrayName>[array_size]

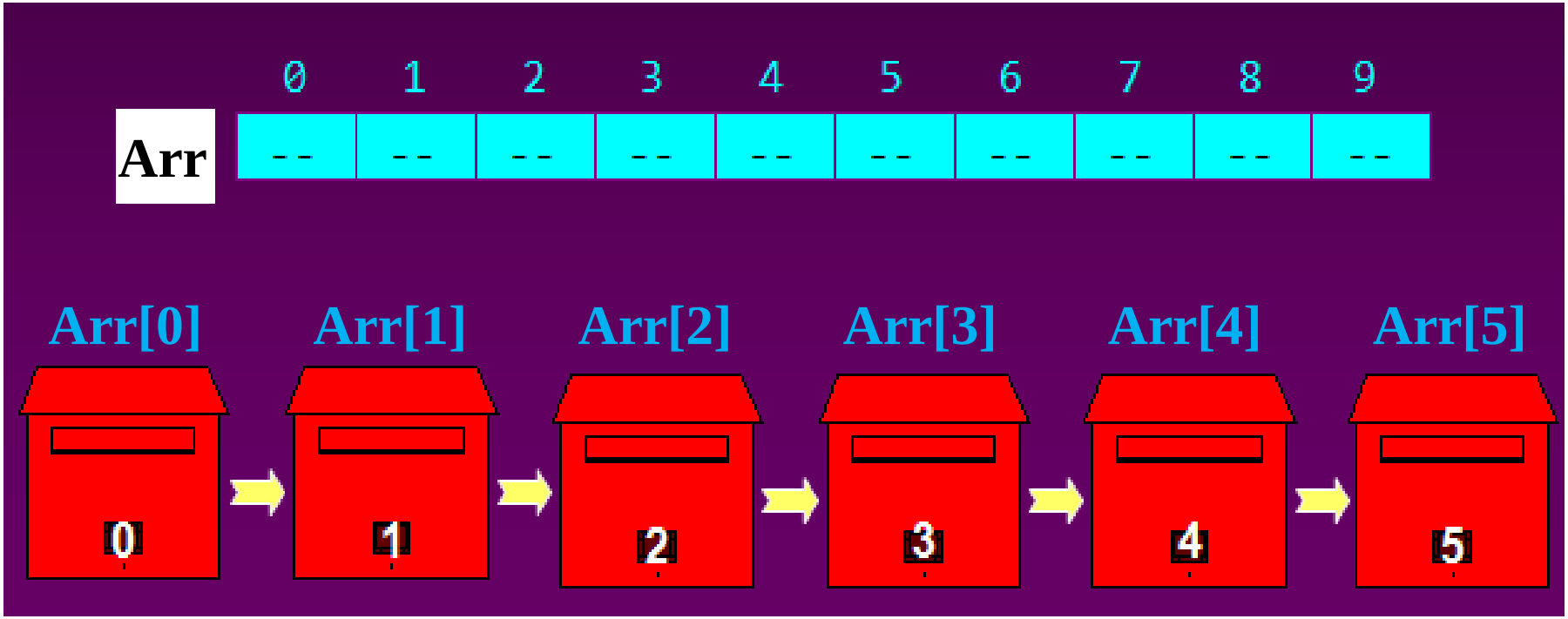
  Ex. int Arr[10];

- The array elements are all values of the type **<type>.**
- The size of the array is indicated by **<array_size>**, the number of elements in the array.
- **The size of the array needs to be predefined.** (We will see dynamically sized arrays later)
- **<array_size>** must be an **integer** constant or a constant expression.

Declaration & Initialization 1D arrays

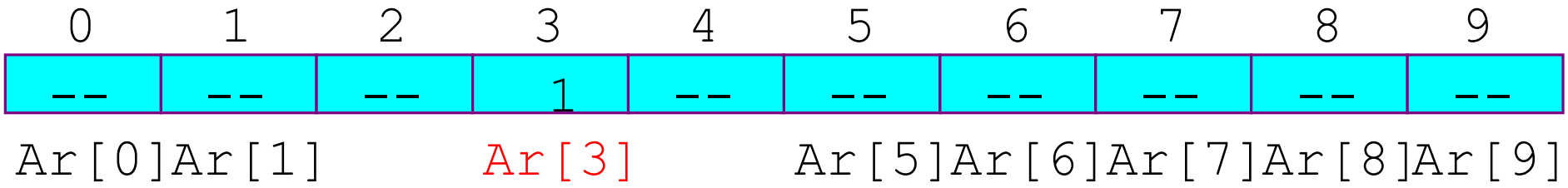
int Arr[10];

Arrays start from index 0.



Declaration & Initialization 1D arrays

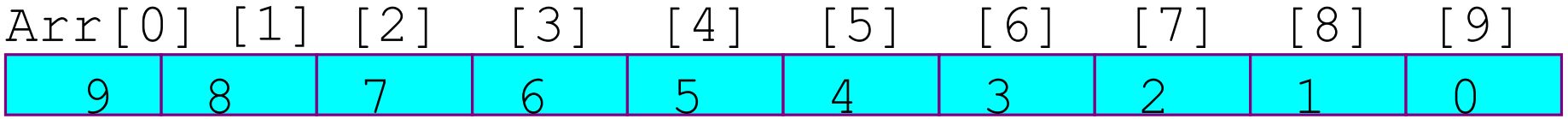
```
Ar[3] = 1;
int x = Ar[3];
```



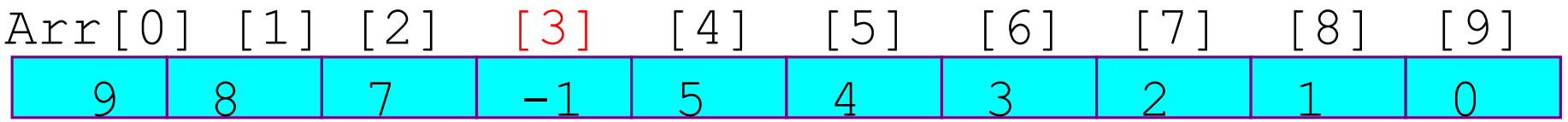
Declaration & Initialization 1D arrays

Compile Time Initialization

- int Arr[10] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};



- Arr[3] = -1;



Compile Time Initialization

- Initializers
  int n[5] = { 1, 2, 3, 4, 5 };
  - If not enough initializers, rightmost elements become **0**
    int n[5] = { 0 } All elements 0
  - If too many a syntax error is produced syntax error
  - C arrays have no bounds checking

- If size omitted, initializers determine it
  int n[ ] = { 1, 2, 3, 4, 5 };
  - 5 initializers, therefore 5 element array

Run Time Initialization

```
void main()
{
        int i, arr[20];
        for(i=0; i<20; i++)
        {
                if(i<10)
                        scanf("%d", &arr[i]);
                else
                        arr[i] = 1234;
        }
}
```

Array Index Syntax Ambiguity

int a[5]={1,2,3,4,5};

printf("%d %d", a[3], 3**[a]**);

3[a] is also valid and both expressions give similar output as 4.

1D Array Programs

```
int Arr[10], i = 7, j = 2, k = 4;
Arr[0] = 1;
Arr[i] = 5;
Arr[j] = Arr[i] + 3;
Arr[j+1] = Arr[i] + Arr[0];
Arr[Arr[j]] = 12;
Scanf("%d",&Arr[k]); //the input value is 3
```

What will be the values at each index of the array?

(Answer on the next slide)

1D Array Programs

- The Answer is…

Write a program for the Output below…

(Program on the next slide)

```
Element         Value         Histogram
      0            19         ********************
      1             3         ***
      2            15         ***************
      3             7         *******
      4            11         ***********
      5             9         *********
      6            13         *************
      7             5         *****
      8            17         *****************
      9             1         *
```

```c
/* Array Example
   Histogram printing program */
#include <stdio.h>
#define SIZE 10

int main()
{
   int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
   int i, j;

   printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );

   for ( i = 0; i <= SIZE - 1; i++ ) {
      printf( "%7d%13d          ", i, n[ i ]) ;

      for ( j = 1; j <= n[ i ]; j++ )    /* print one bar */
         printf( "%c", '*' );

      printf( "\n" );
   }

   return 0;
```

Caution! Caution! Caution!

- It is the programmer's responsibility to avoid indexing off the end of an array
  - *Likely* to corrupt data
  - May cause a *segmentation fault*
  - Could expose system to a *security hole!*
- *C* does NOT check *array bounds*
  - I.e., whether index points to an element within the array
  - Might be high (beyond the end) or negative (before the array starts)

Searching and Sorting

- Searching for elements within an Array (finding the index of a particular element) and Sorting the elements of an Array in some order are two of the most primary operations one can perform on an Array.

- Computer Scientists have done a lot of research on Searching and Sorting Algorithms.

- Besides these two operations other operations that can be performed are: Deleting an element of the array and Inserting a new element in the array

Searching and Sorting

Sorting Algorithms

- Bubble Sort
- Insertion Sort
- Selection Sort
- Merge Sort
- Shell Sort
- And many more sorting algorithms…

Searching Algorithms

- Linear Search
- Binary Search

2D Arrays

- The arrays we have discussed so far are known as one-dimensional arrays because the data are organized linearly in only one direction.

- Many applications require that data be stored in more than one dimension.

- One common example is a table, which is an array that consists of rows and columns.

2D Arrays

- The same reason that necessitated the introduction of 1-D array can also be extended to Multi-dimensional Array.
- For example, to store the grades of (30) students, each of which is taking a number of courses (5 say), we would either use 30 1-D arrays, one for each student or 5 1-D arrays, one for each course
- Multi-Dimensional array allows us to handle all these using a single identifier.
- 2-Dimensional array is the most commonly used multi-dimensional arrays.
- Other multi-dimensional arrays are also possible.

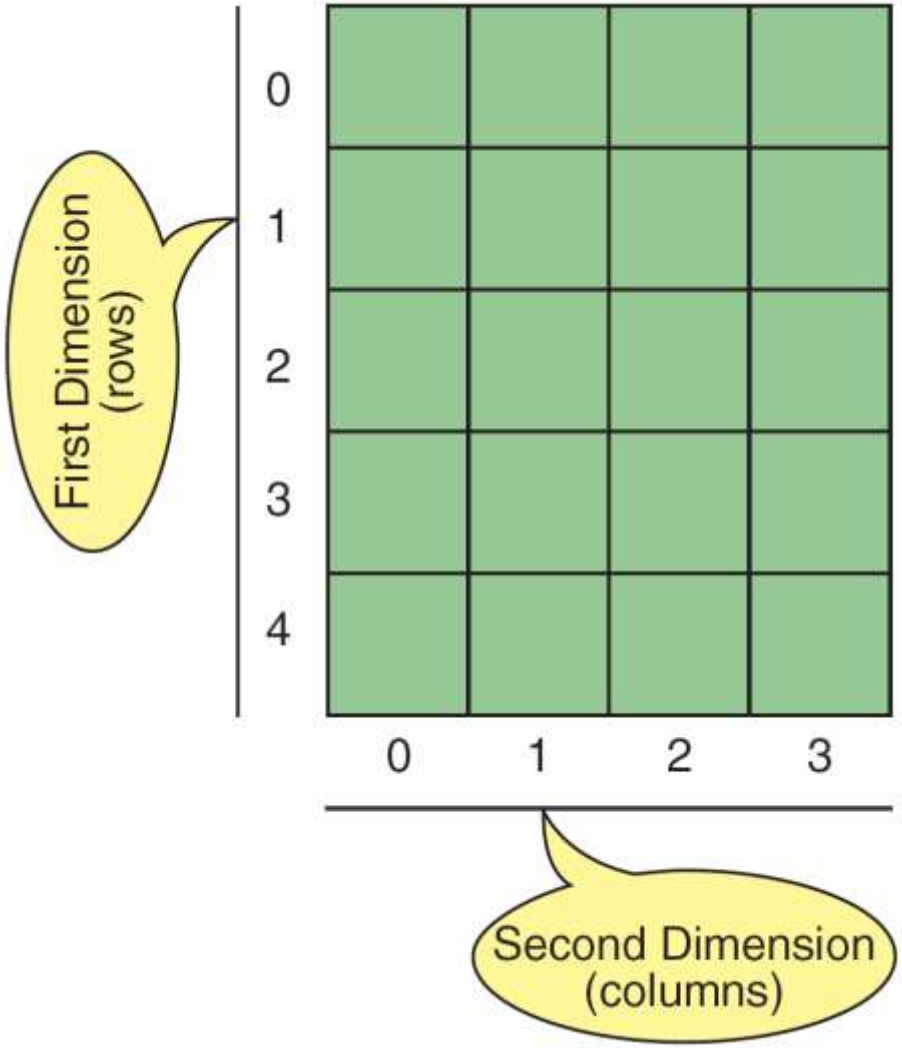
2D Arrays

- A two-dimensional array is declared in a similar manner to 1-D array but with the addition of one more bracket as in the following example;

  int arr[3][4];

- Although it is convenient to think of a 2-D array as a table as will be shown in the next slide, in reality, **arr** is simply an array of arrays**.**

2D Arrays



2D Arrays

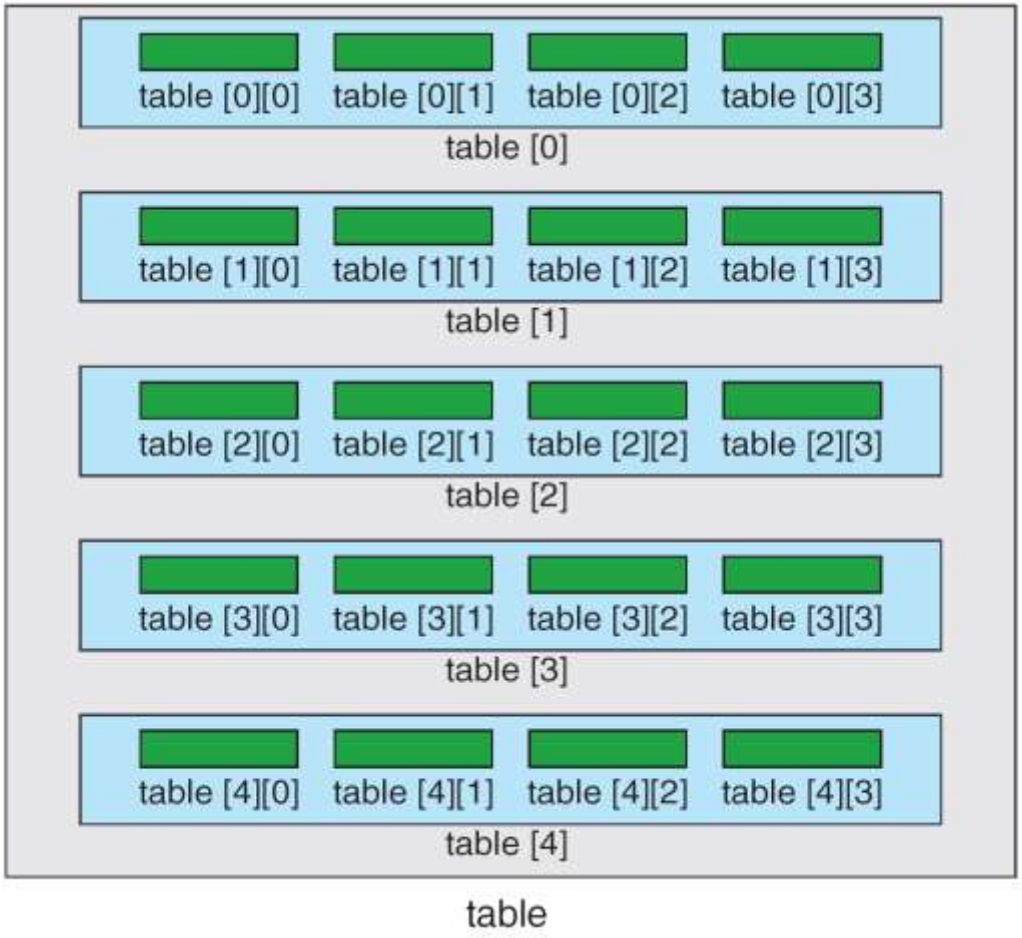
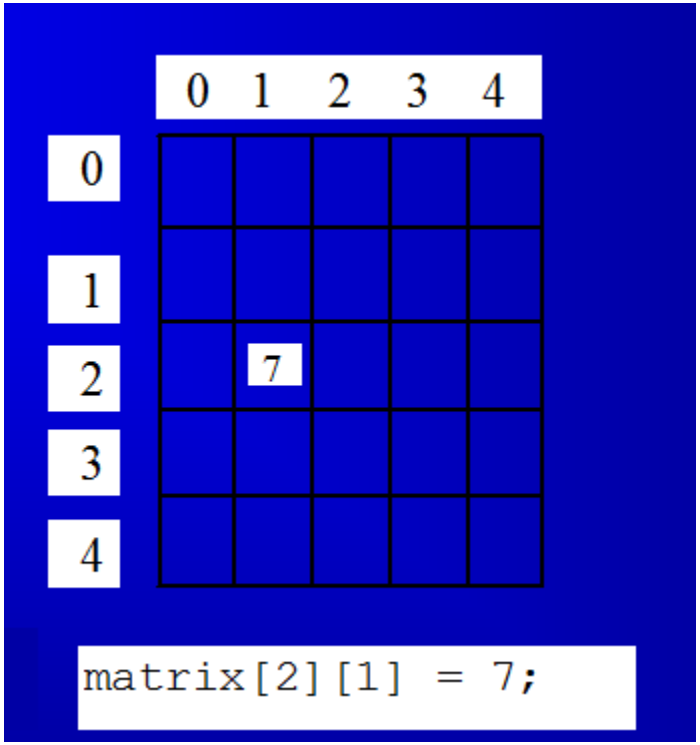


table [0][0] table [0][1] table [0][2] table [0][3]
table [0]
table [1][0] table [1][1] table [1][2] table [1][3]
table [1]
table [2][0] table [2][1] table [2][2] table [2][3]
table [2]
table [3][0] table [3][1] table [3][2] table [3][3]
table [3]
table [4][0] table [4][1] table [4][2] table [4][3]
table [4]
table


0 1 2 3 4
0
1
2
3
4
7
matrix[2][1] = 7;

2D Arrays Initialization

- Similar to 1-D array, a two-dimensional array can be created as a list of array initializers, one for each row in the array, as shown by the following:

```
int arr[ ][ ] =  { {  1,  0, 12, -1 }, {  7, -3,  2,  5 }, { -5, -2,  2, 9}};
```

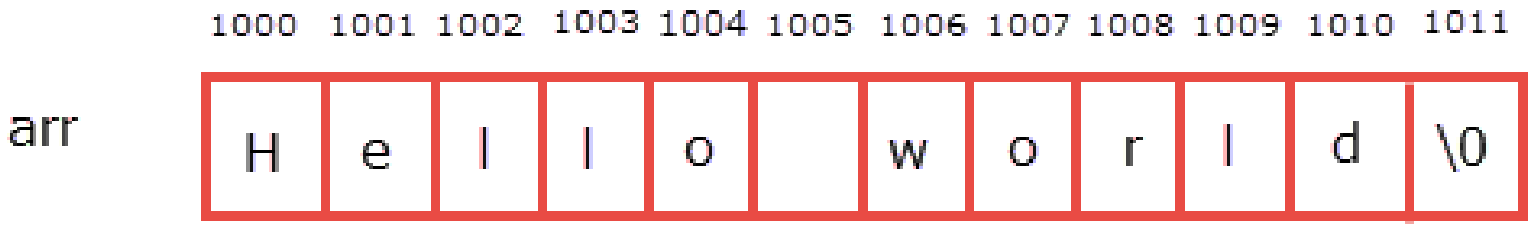
OR

```
Int arr[3][4] = {1, 0, 12, -1, 7, -3, 2, 5, -5, -2, 2, 9}
```

- And you can also perform runtime initialization using nested for loops

Memory Allocation

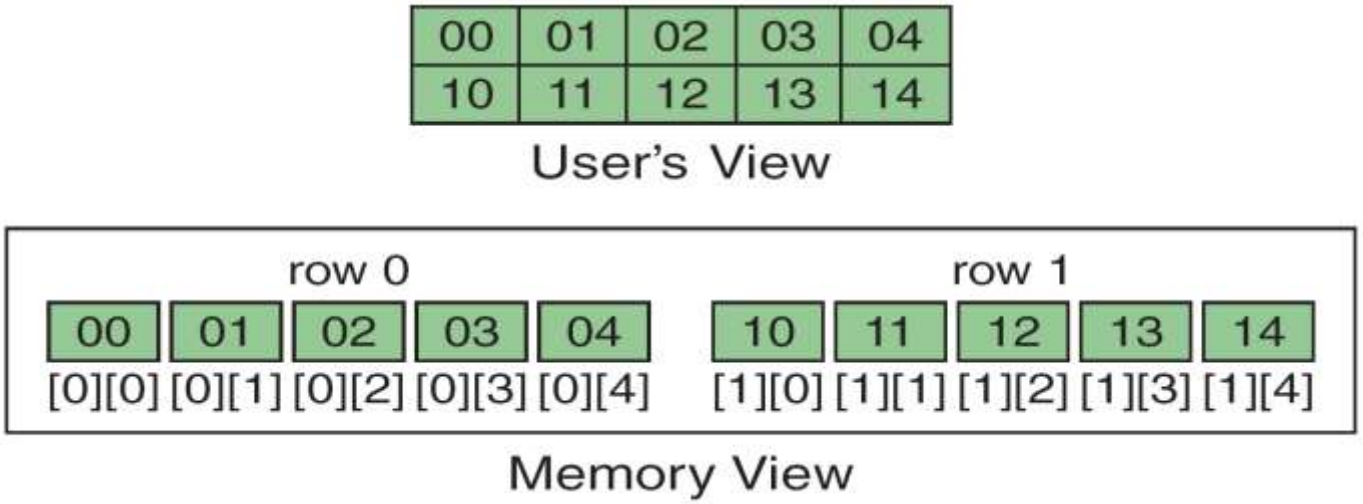
- If the address of arr[0] is 1000, then the address of arr[1] will be 1002, because int is 2 bytes long. Address of arr[2] will be 1004, arr[3] will be 1006 and so on…



12 bytes of memory is allocated to store 12 characters

Memory Allocation

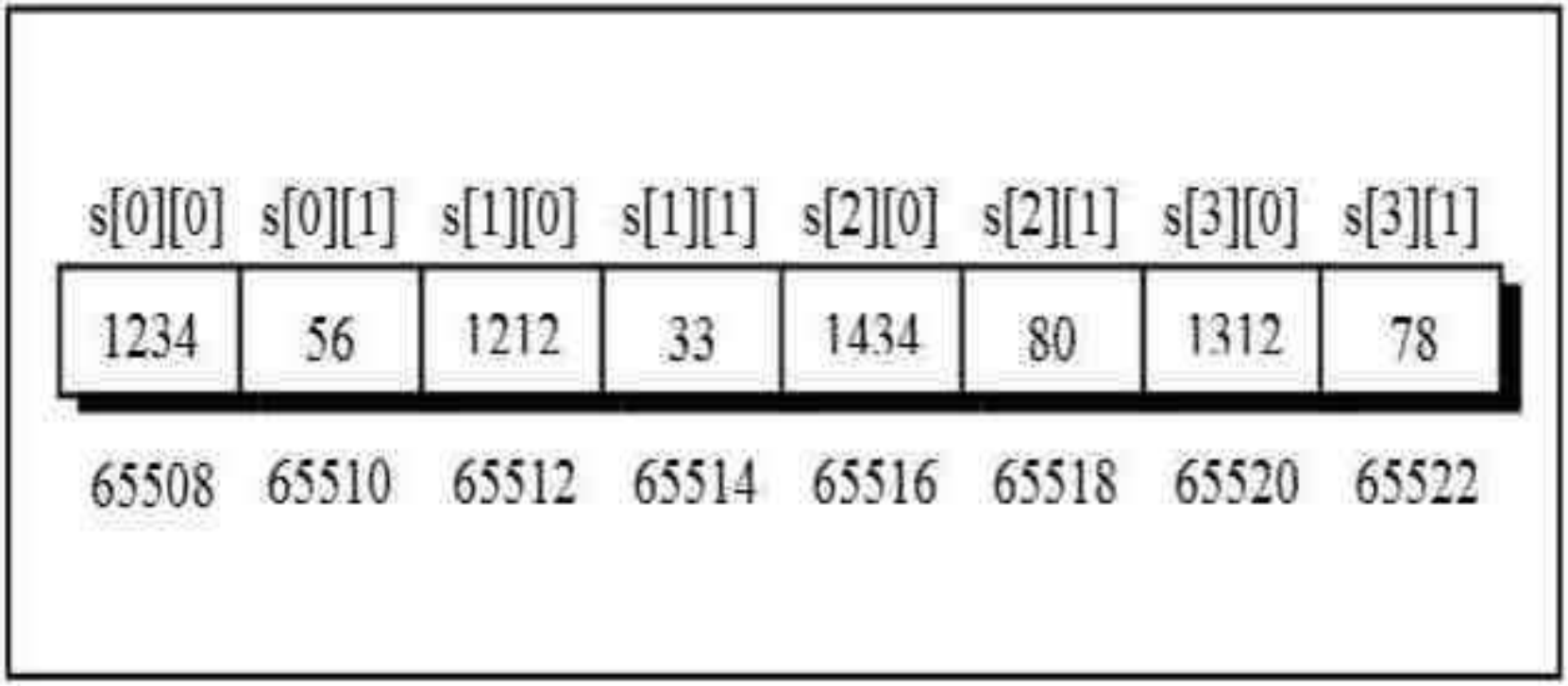
- Even for two dimensional Arrays, memory is allocated in a continuous manner and not like a table inside the RAM



Memory Allocation

If we declare a 2D array as: int s[4][2];

And address of s[0][0] is 65508



2D Array Example

```
1  /* This program changes a two-dimensional array to the
2     corresponding one-dimensional array.
3        Written by:
4        Date:
5  */
6  #include <stdio.h>
7  #define ROWS 2
8  #define COLS 5
9
10 int main (void)
11 {
12 // Local Declarations
13    int table [ROWS] [COLS] =
14       {
15        {00, 01, 02, 03, 04},
16        {10, 11, 12, 13, 14}
17       }; // table
18    int line [ROWS * COLS];
```

2D Array Example (Continued)

```
20  // Statements
21     for (int row = 0; row < ROWS; row++)
22        for (int column = 0; column < COLS; column++)
23           line[row * COLS + column] = table[row][column];
24
25     for (int row = 0; row < ROWS * COLS; row++)
26        printf(" %02d ", line[row]);
27
28     return 0;
29  }  // main
```

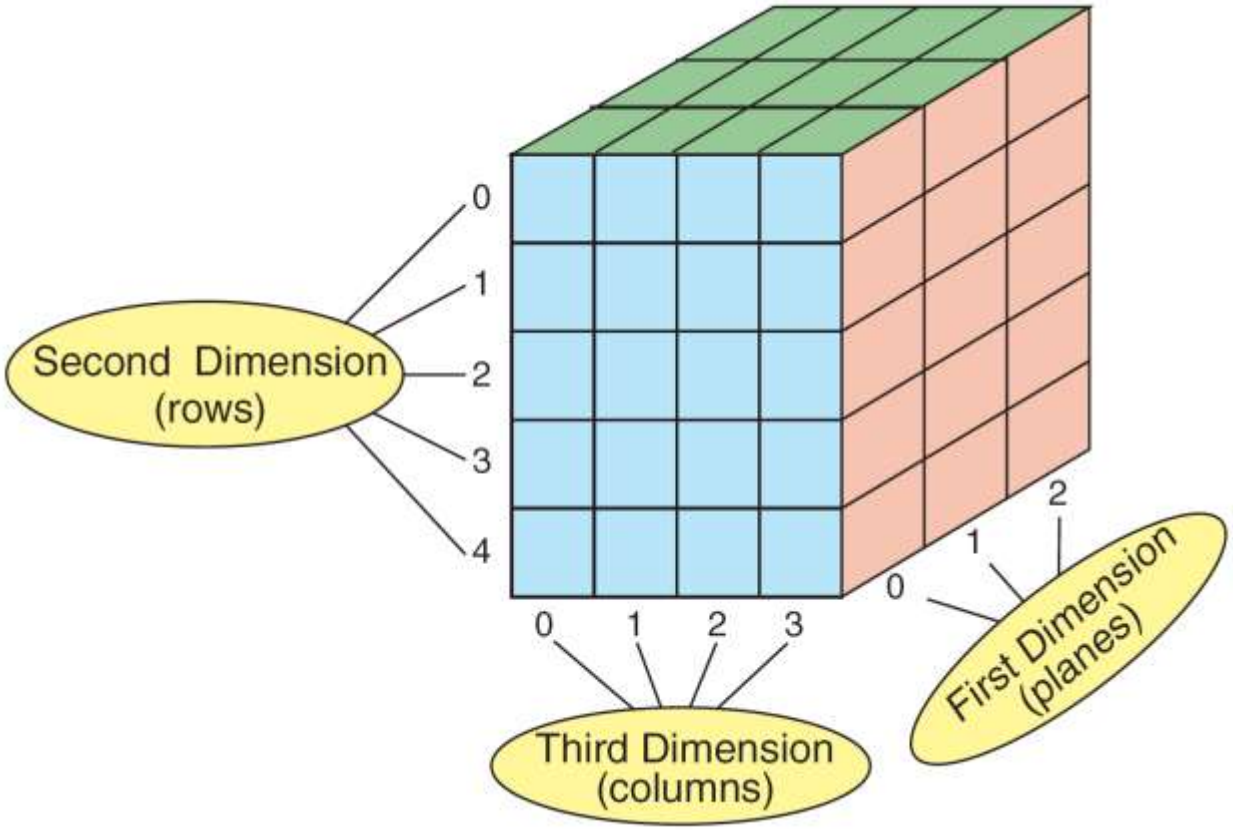
```
Results:
00   01   02   03   04   10   11   12   13   14
```

Multi-Dimensional Arrays (Topic not part of Syllabus)

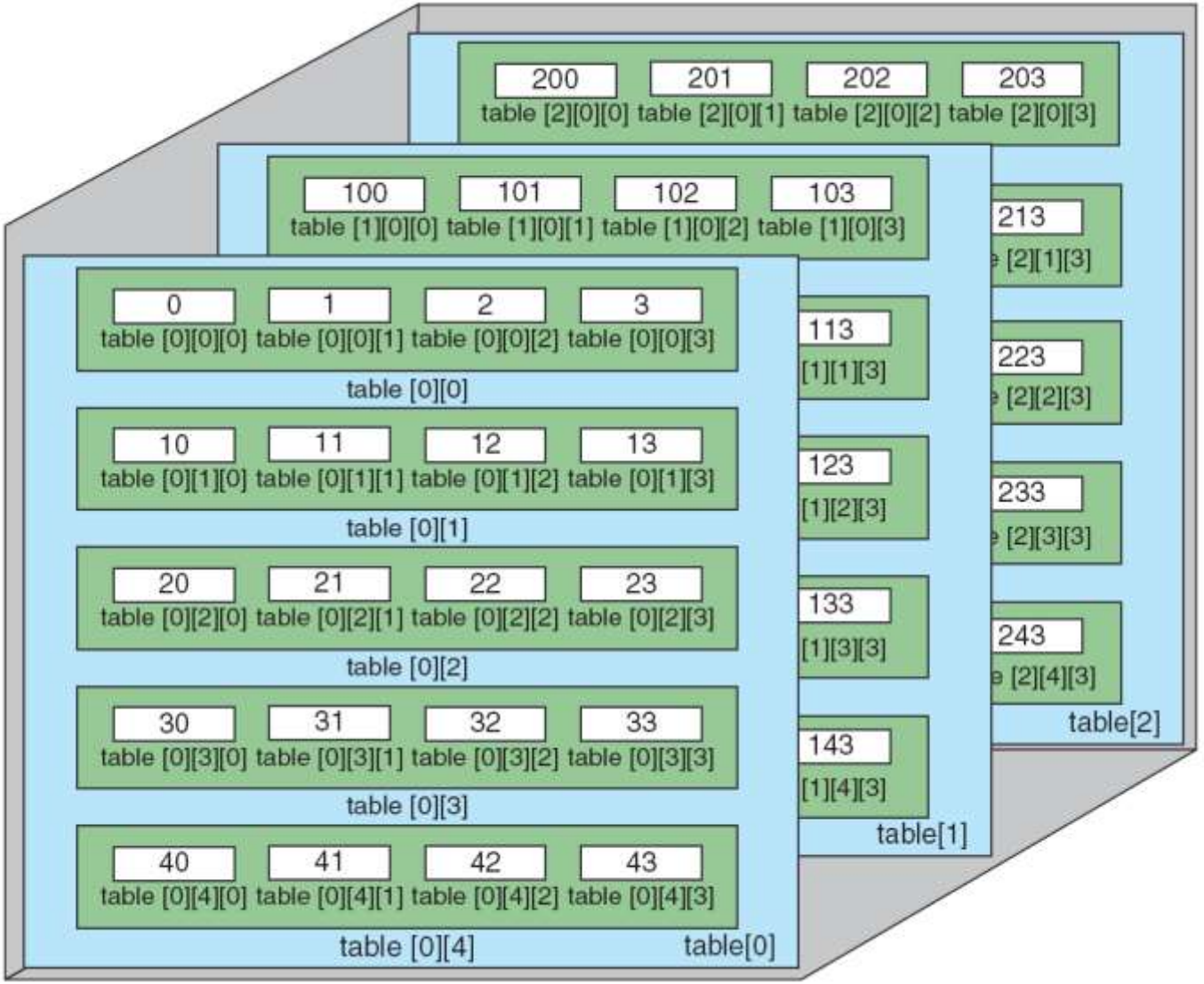
int arr[3][5][2][4];
float num[2][7][3];

- Multidimensional arrays can have three, four, or more dimensions.
- The first dimension is called a plane, which consists of rows and columns.
- The C language considers the three-dimensional array to be an array of two-dimensional arrays.

Multi-Dimensional Arrays (Topic not part of Syllabus)



Multi-Dimensional Arrays



Dynamic Arrays

- In the programs seen in previous slides, all memory needs were determined before program execution by defining the variables needed.
- **The size of the array needs to be predefined. Such arrays are called Static arrays.**
- But there may be cases where the memory needs of a program can only be determined during runtime. For example, when the memory needed depends on user input.
- In these cases, programs need to dynamically allocate memory, for which the C language integrates the operators malloc, calloc and realloc.

Previous year Questions

Fill in the Blanks :

1. An array can be initialized either at compile time or at _____ .

**2. An array created using malloc() at run time is referred to as _____ array.

3. The number used to refer to a particular element of an array is called it's _____.

4. Int c[3][3]={1,2,3,4,5,6,7,8,9} _____ will be the value of c[1][1].

5. Arrays always occupy _____ memory.

Previous year Questions and other questions

State True or False :

1. An array is a fixed sized sequenced collection of elements of same data types.

2. It is necessary to have initialization when the size of the dimension is kept blank in array.

3. The first dimension refers to column number and the second dimension refers to the row number in a two dimensional array.

4. For the initialization, int arr[4] = {1,2,6}; the elements of the array will have the values 1,2,6 and 0.

5. When we place a semi colon after the for statement , the compiler will generate an error message .

Previous year Questions

Questions :

1. Write a program to read two 3 X 3 matrices from the user and store the addition of the two in a third matrix. i.e. c[3][3] = a[3][3] + b[3][3]

2. Define Array. Explain Types of Arrays.

3. What is the limitation of an Array? Write a program to multiply two 2 X 2 matrices.

4. If an array is declared as float x[5][3]; then what will be address of x[2][2] if the

address of the first element is 1000? Explain with memory layout.

5. What happens when an array with a specified size is assigned at compile time,

a. with values less than the specified size.

b. with values more than the specified size.